Many concepts and two logics of algorithmic reduction

Giorgi Japaridze

Institute of Artificial Intelligence, Xiamen University and
Department of Computing Sciences, Villanova University

Abstract

Within the program of finding axiomatizations for various parts of *computability logic*, it was proven earlier that the logic of interactive Turing reduction is exactly the implicative fragment of Heyting's intuitionistic calculus. That sort of reduction permits unlimited reusage of the computational resource represented by the antecedent. An at least equally basic and natural sort of algorithmic reduction, however, is the one that does not allow such reusage. The present article shows that turning the logic of the first sort of reduction into the logic of the second sort of reduction takes nothing more than just deleting the contraction rule from its Gentzen-style axiomatization. The first (Turing) sort of interactive reduction is also shown to come in three natural versions. While those three versions are very different from each other, their logical behaviors (in isolation) turn out to be indistinguishable, with that common behavior being precisely captured by implicative intuitionistic logic. Among the other contributions of the present article is an informal introduction of a series of new — finite and bounded — versions of recurrence operations and the associated reduction operations.

MSC: primary: 03B47; secondary: 03F50; 03B70; 68Q10; 68T27; 68T30; 91A05
Keywords: Computability logic; Intuitionistic logic; Affine logic; Linear logic; Interactive computation; Game semantics.

1 Introduction

This article is a new addition to the evolving list of papers [7, 8, 9, 10, 12, 13, 14, 15, 16, 18, 19, 20] devoted to finding axiomatizations for various fragments of *computability logic*. The latter is a program for redeveloping logic as a formal theory of computability, as opposed to a formal theory of truth which it has more traditionally been.

Under the approach of computability logic, formulas express interactive computational problems defined as games between the two players $\top$ (*machine*) and $\bot$ (*environment*), with logical operators standing for basic operations on games. "Truth" of a problem/game means existence of an algorithmic solution, i.e. $\top$'s effective winning strategy. And validity of a logical formula is understood as (such) truth under every particular interpretation of atoms. With this semantics, computability logic provides a systematic answer to the fundamental question "*what can be computed?*", just as classical logic is a systematic tool for telling what is true. Furthermore, as it turns out, in positive cases "*what* can be computed" always allows itself to be replaced by "*how* can be computed", which makes computability logic of potential interest in not only theoretical computer science, but many more applied areas as well, including interactive knowledge base systems, resource oriented systems for planning and action, or declarative programming languages. On the logical side, computability logic can serve as a constructive and computationally meaningful alternative to classical logic as a basis for applied theories. The first concrete steps in the direction of materializing this potential have been made very recently in [19], where a computability-logic-based system of arithmetic was constructed — a formal theory whose every formula expresses a computational problem and every proof encodes an algorithmic solution for such a problem, thus fully reducing problem-solving to theorem-proving.

Having said the above, motivationally or technically (re)introducing computability logic is not within the scope of the present paper. This job has been done in [6, 11, 17], and the present paper, whose goal is merely putting one more brick into the foundation of the edifice under construction, primarily targets readers already familiar with the basics of computability logic. Yet, as it happens, the proof of the main technical result of the paper, given in Sections 2-4, can be understood in full detail without knowing much (if anything at all) about computability logic. Those with no prior acquaintance with the subject may benefit from browsing the rest of the paper just as well. Even though doing so would be certainly insufficient for getting full insights into the project, chances are that such a reader may at least start feeling curious enough to be willing to look at some additional literature. The most recommended reading for familiarity with the basic philosophy, motivations, concepts and techniques of computability logic is the tutorial-style [17].

Here we very quickly review, in a simplified form, certain basic concepts on the games used in computability logic, to refresh the memory of those previously exposed to the subject, and to provide some clues to those who have never seen it.

A **move** means a finite string over some fixed alphabet, such as the standard keyboard alphabet. A **labeled move** is a move prefixed with $\top$ or $\bot$. The meaning of such a prefix ("*label*") is to indicate which of the two players has made the move. A **run** is a (finite or infinite) sequence of labeled moves, and a **position** is a finite run. Runs (and positions as special cases of runs) are thus records of interaction histories, spelling out what moves, in what order and by which players have been made during a given play of a game.

A **game**[1] is a pair $(\mathbf{Lr}, \mathbf{Wn})$ consisting of what are called its **structure** ($\mathbf{Lr}$) and **content** ($\mathbf{Wn}$). One of the many equivalent ways to define the structure component of a game is to say that it is a binary relation between positions and labeled moves. Then the intuitive meaning of $\mathbf{Lr}(\Phi, \wp\alpha)$ is that α is a **legal move** by player $\wp$ in position Φ. A run where all moves are legal (in the positions preceding those moves) is said to be a **legal run**. The empty run is thus always trivially legal. As expected, "**illegal**", whether it be a move or a run, means "not legal". As for the content $\mathbf{Wn}$ of a game, it can be defined as a set of legal runs, whose elements are said to be (and intuitively thought of as) the runs **won** by player $\top$ (and hence lost by $\bot$), with all other legal runs considered **lost** by $\top$ (and hence won by $\bot$). As for illegal runs, they are always considered to be lost by the player who made the first illegal move.

Note the relaxed nature of such games. There are no conditions on the order in which moves should or could be made (such as, say, strict alternation of players' turns), and generally either player may have legal moves in a given position/situation. This makes the games of computability logic a rather direct (without any "bureaucratic pollutants") and flexible tool for modeling interaction, including asynchronous interactions. The relaxed nature of our games makes it impossible to understand game-playing strategies as functions from positions to moves, as this is typical for most other game models. Instead, ($\top$'s effective) strategies are understood as interactive machines. Such a machine is nothing but a Turing machine with the additional capability of making moves. The adversary can also move at any time, with such moves being the only nondeterministic events from the machine's perspective. The play is fully visible to the machine through an additional, read-only *run tape* which, at any time, spells the "current position" of the play. We say that such a machine **wins** a given game iff, no matter how the adversary acts (what moves it makes and when it makes them), the run incrementally spelled on the run tape is won by $\top$.

A universal-utility game semantics should be about interaction, whereas functions are inherently non-interactive. The above-mentioned traditional, *strategies-as-functions*, approach misses this important point and creates a hybrid of interactive (games) and non-interactive (functions) entities. To see the resulting loss, it would be sufficient to reflect on the behavior of one's personal computer. The job of your computer is to play one long — potentially infinite — game against you. Now, have you noticed your "adversary" getting slower every time you use it? Probably not. That is because the computer is smart enough to follow a non-functional strategy in this game. If

[1] To what we refer as a "game" in this paper, is in fact called a "constant game" in computability logic, and the term "game" is reserved for a slightly more general concept. Considering only constant games is sufficient for our present purposes though and, to keep things simple, we are using the term "game" for them.

its strategy was a function from positions (interaction histories) to moves, the response time would inevitably keep worsening due to the need to read the entire — continuously lengthening and, in fact, practically infinite — interaction history every time before responding. Defining strategies as functions of only the latest moves (rather than entire interaction histories) in Abramsky and Jagadeesan's [1] tradition is also not a way out, as typically more than just the last move matters. Back to your personal computer, its actions certainly depend on more than your last keystroke. Thus, the difference between the traditional *functional* strategies and the *post-functional* strategies of computability logic is not just a matter of taste or convenience. It will become especially important when it comes to (yet to be developed) interactive complexity theory: hardly any meaningful interactive complexity theory can be done with the strategies-as-functions approach. And complexity issues will inevitably come forward when computability logic or similar approaches achieve a certain degree of maturity: nowadays, 95% of the theory of computation is about complexity rather than just computability. Time has not yet matured for seriously addressing complexity issues within the framework of computability logic though, and the latter, including the present paper, continues to be focused on just computability, which still abounds with open questions waiting for answers.

In the above outline, we described interactive Turing machines in a relaxed fashion, leaving to the reader filling technical details about, say, how, exactly, moves are made by the machine, how many moves either player can make at once, what happens if both players attempt to move "simultaneously", etc. As it turns out, all reasonable design choices yield the same class of winnable games as long as we consider a certain natural subclass of games called **static**. Such games are obtained by imposing a certain simple formal condition on games (see, e.g., Section 5 of [17]), which we do not reproduce here as nothing in this paper relies on it. We will only point out that, intuitively, static games are interactive tasks where the relative speeds of the players are irrelevant, as it never hurts a player to postpone making moves. In other words, static games are the games that are contests of intellect rather than contests of speed. And one of the theses that computability logic philosophically relies on is that static games present an adequate formal counterpart of our intuitive concept of "pure", speed-independent interactive computational problems. Correspondingly, computability logic restricts its attention (more specifically, possible interpretations of the atoms of its formal language) to static games. Needless to say, the class of static games is closed under all game operations studied in computability logic.

Among the most interesting of such operations are several versions of *reduction*. The simplest form of reduction (of B to A) is $A \rightarrow B$. This is a parallel play of the two games A and B with the roles of $\top$ and $\bot$ interchanged in the antecedent. Winning a given run of this game for $\top$ means that whenever the adversary wins A, $\top$ has to win B.

More formally, every legal move of $A \rightarrow B$ has to be prefixed with one of the two strings "0." or "1." to indicate in which of the two components the move is made. The effect of a move $0.\alpha$ is making move α in the antecedent, and the effect of $1.\alpha$ is making move α in the consequent. In order for such a move $0.\alpha$ or $1.\alpha$ to be legal, α should be a legal move in (the corresponding position of) the corresponding component A or B. Then a legal run Γ of $A \rightarrow B$ is considered won by $\top$ iff $\Gamma^{1.}$ is a $\top$-won run of B or $\neg\Gamma^{0.}$ is a $\bot$-won run of A. Here $\Gamma^{1.}$ means the result of deleting from Γ all moves except those of the form $1.\alpha$, and then further deleting the prefix "1." in such moves. Similarly for $\Gamma^{0.}$. And $\neg\Gamma^{0.}$ means the result of turning upside down (so that $\top$ becomes $\bot$ and vice versa) all labels in $\Gamma^{0.}$.

As can be felt from the above passage, formal definitions may not be as nice to work with as informal or intuitive explanations. For this reason, our subsequent explanations of game operations in this section will be limited to informal ones, keeping in mind that they certainly can be turned into strict technical definitions.

Since the roles of the players are switched in the antecedent of $A \rightarrow B$, the A component, as a computational problem from $\bot$'s perspective, becomes a computational resource for $\top$. Namely, $\top$ can observe how the adversary is solving/playing (a single session) of A, and utilize that information in its own solving/playing B. The following example illustrates the above-said. Let $\mathbf{H}$ be the *halting problem*, which can be understood as a game of depth 2 (i.e., no legal run has more than two moves).

In the initial (empty) position of this game, only $\bot$ has legal moves, and such a move should be the phrase "Does Turing machine m halt on input i?", where m is a legitimate description of a Turing machine and i a possible input for it. After such a move is made, the second and last legal move is by $\top$, which should be either "Yes" or "No". $\top$ wins iff it correctly answers the question asked by $\bot$. The failure by $\bot$ to make an initial move is considered $\top$'s win, as there was no question to answer. And, if such a move is made, then the failure of $\top$ to respond is considered $\bot$'s win. The *acceptance problem* **A** is similar, only it is about whether a given machine accepts (rather than halts on) a given input. Neither **H** nor **A** is decidable, which obviously means that these problems, as games, have no algorithmic winning strategies. However, **A** is algorithmically reducible to **H**. Specifically, $\top$ *does* have an effective winning strategy in the game $\mathbf{H} \rightarrow \mathbf{A}$, which goes like this. Wait till, in the consequent, $\bot$ asks a question regarding whether a certain machine m accepts a certain input i. Then, in the antecedent, ask a counterquestion regarding whether m halts on i (the same m and i). If an answer to this counterquestion is "No", answer "No" to the original question in the consequent and rest your case, as not halting implies not accepting. Otherwise, if the answer in the antecedent is "Yes", simulate machine m on input i until it halts, and say "Yes" or "No" in the consequent depending on whether the simulation accepted or rejected. (Of course, the possibility that the simulation goes on forever is not ruled out here; but this would mean that m does not really halt on i, and having lied in the antecedent would make $\bot$ lose the game regardless of what happens in the consequent).

In fact, $\rightarrow$ is not only the simplest but also the strongest form of reduction. In this respect, at the other extreme is the weakest reduction $\circ\!\!-$. The game $A \circ\!\!- B$ can be characterized in the same intuitive terms as $A \rightarrow B$, with the difference that, in $A \circ\!\!- B$, unlike $A \rightarrow B$, $\top$ is allowed to reuse A (as a computational resource) any number of times, with "reuse" here understood in the strongest algorithmic sense possible. Namely, at any time, $\top$ can temporarily abandon a given position of A (while reserving the right to come back to it later), backtrack to any of the earlier positions of it and try a different continuation from there, thus forcing $\bot$ to play multiple parallel sessions of A against such a capricious adversary in this most unfair game: the failure of $\bot$ to win A in *all* sessions of it automatically results in $\top$'s victory.

A while ago we saw how to reduce the acceptance problem to the halting problem in the strong sense of $\rightarrow$. We would not have been just as successful if instead of the acceptance problem **A** we had taken the *Kolmogorov complexity* problem **K**, where the initial move of the form "What is the Kolmogorov complexity of number n?" is by the environment, obligating the machine to respond with a move "m" such that m is (indeed) the Kolmogorov complexity of n, i.e., m is the size of the smallest Turing machine that returns n on input 0. One can show that, unlike $\mathbf{H} \rightarrow \mathbf{A}$, the game $\mathbf{H} \rightarrow \mathbf{K}$ does not have an algorithmic winning strategy. But the weaker game $\mathbf{H} \circ\!\!- \mathbf{K}$ certainly does, due to the fact that, in it, the reduction is allowed to use the antecedent repeatedly. Such a strategy goes like this. Wait to hear a question about the Kolmogorov complexity of a number n in the consequent. Then, starting from $m = 0$, do the following. Duplicate the original antecedent and save one copy of it for future usage (further duplications). In the other copy, ask the counterquestion regarding whether the machine (encoded by) m halts on input 0. If you hear "No", increment m to $m + 1$ and repeat the step. Otherwise, if you hear "Yes", simulate m on input 0; if the simulation shows that m returns n on input 0, answer $|m|$ (where $|m|$ is the size of m) to the original question in the consequent, and wash your hands. In any other case, increment m to $m + 1$ and repeat the step.

There is a whole spectrum of natural reduction operations of intermediate strength between $\rightarrow$ and $\circ\!\!-$. Only some of those have been officially introduced within the framework of computability logic so far, with more to be probably defined later depending on particular needs, motivations and tastes. It has been repeatedly pointed out earlier that the formalism of computability logic is open-ended, welcoming any meaningful augmentations.

Among the most natural and simple reduction operations of intermediate strength is $\succ\!\!-$. Just like $A \circ\!\!- B$ and unlike $A \rightarrow B$, the game $A \succ\!\!- B$ allows $\top$ to reuse A infinitely many times. But the form of reusage is less flexible here: $\top$ is required to restart A from the very beginning every time it wants to reuse it, meaning that it essentially cannot utilize the advantages of backtracking

permitted in $A \circ\!\!- B$. Specifically, unless $\bot$ plays in exactly the same ways in different parallel sessions of A, $\top$ has no possibility to experiment with different reactions to the same actions by the adversary.

Thus, the difference between $A \succ\!\!- B$ and $A \circ\!\!- B$ is in the allowed *type* of reusage of A, with the *quantity* of reusages being otherwise unlimited. Yet, as it happens, this difference in the types of reusage automatically yields a difference in the quantities as well. Specifically, in $A \succ\!\!- B$ at most countably many parallel runs of A can be generated, while in $A \circ\!\!- B$, when A has infinitely long legal runs, that quantity can be a continuum. A very simple modification in the formal definition of $\circ\!\!-$, given later in Section 5, turns it into a definition of the Blass-style [2] reduction $\circ\!\!-^{\aleph_0}$, by its strength strictly between $\succ\!\!-$ and $\circ\!\!-$. The type of reusage of A in $A \circ\!\!-^{\aleph_0} B$ is the same as in $A \circ\!\!- B$, but the quantity of reusages is limited to the countably infinite cardinal $\aleph_0$.

The operation $\circ\!\!-^{\aleph_0}$ is apparently the weakest nontrivial strengthening of $\circ\!\!-$. Both $\circ\!\!-$ and $\succ\!\!-$ can be further strengthened to $\circ\!\!-^F$ and $\succ\!\!-^F$ by allowing $\top$ to reuse the antecedent only a finite (yet unbounded) number of times. In turn, the operations $\circ\!\!-^F$ and $\succ\!\!-^F$ can be further strengthened to bounded versions of $\circ\!\!-$, $\succ\!\!-$. The simplest form of a bounded version of $\supset \in \{ \succ\!\!-, \circ\!\!- \}$ would be $\supset^n$, where n is a natural number. It means the same as $\supset$, only the number of allowed (re)usages of the antecedent is limited to n, so that $A \supset^0 B$ is nothing but simply B, and $A \supset^1 B$ is nothing but $A \to B$. But bounds do not necessarily have to be natural numbers. Reasonable transfinite ordinals could be interesting to study as well, such as ordinals less than ϵ_0. For example, where ω is the smallest infinite ordinal, $A \supset^\omega B$ would mean a game where $\top$ has to declare a number n before starting using A, after which the game continues as if it was $A \supset^n B$. This generalizes to $A \supset^{k\omega} B$ for any $k \geq 0$, where k (rather than just one) declarations $n_1, \ldots, n_k$ are made. The first declaration n_1 opens n_1 copies of A for usage; the second declaration n_2, which can be made any time later when the previously "activated" copies of A are perhaps already at advanced stages, creates the possibility to use n_2 additional copies; the third declaration activates n_3 additional copies, etc., with the overall number of (re)usages of A thus eventually not exceeding the finite $n_1 + \ldots + n_k$. Next, $A \supset^{\omega^2} B$ would be a game where $\top$ has to declare a number n before starting using A, after which the play continues as it would proceed in $A \supset^{n\omega} B$. This can be further generalized to $A \supset^{\omega^k} B$ for any $k \geq 0$. Then $A \supset^{\omega^\omega} B$ could be characterized as a game where $\top$'s initial choice of n turns it into a game that proceeds as $A \supset^{\omega^n} B$. And so on and so on.

Furthermore, $\circ\!\!-$ has an even greater variety of bounded versions of potential interest, especially in the (yet to be developed) area of interactive computational complexity theory. One may want to differentiate between just bounds on the overall number of reusages of the antecedent and bounds on, say, the "depths" of reusages. Roughly, the depth of reusages here means the maximum number of ancestor positions of any given run of the antecedent at which restarts ("forkings", "replications") happened. In more precise terms — for those familiar with the relevant formal definitions — such bounds would mean bounds on the heights of the corresponding underlying bitstring trees (see [17]). For $\succ\!\!-$, on the other hand, the above concept of depth is not meaningful as it automatically trivializes to 1 (or to 0, depending on whether or not only proper reusages count).

Finite or bounded versions of reduction operations, except the "most finite" and "most bounded" $\to$, have never been studied, and at this point we do not know what logics they induce. In what follows our focus is only on $\to$, $\succ\!\!-$, $\circ\!\!-^{\aleph_0}$, $\circ\!\!-$.

Of these four operations, $\circ\!\!-$ stands out as, in a sense, most natural and important. What makes $\circ\!\!-$ special is that it has good claims to precisely capture everything that anyone would ever call (interactive) algorithmic reduction. That is in the same sense as Turing computability of functions captures our intuitive concept of effectiveness. What also makes $\circ\!\!-$ natural is that, as suggested by the above characterizations, definitions of other reductions can be easily obtained from the definition of $\circ\!\!-$ by imposing corresponding restrictions on the form and quantity of reusage of the antecedent, with $\to$ being the most extreme nontrivial case, where any proper reusage is simply forbidden altogether.

Alternatively, we can consider $\to$ rather than $\circ\!\!-$ as the basic sort of reduction, and define all weaker versions of reduction in terms of $\to$ and what are called *recurrence operations* ($\downarrow, \downarrow^{\aleph_0}, \downarrow, \ldots$),

in their general spirit resembling the storage operator ! of linear logic. This is exactly the approach that computability logic has prefered to take so far.[2] For instance, [17] treats $A \succ B$ and $A \circ\!\!- B$ as abbreviations of $\barwedge A \to B$ and $\bigcirc\!\!\!\!\downarrow A \to B$, respectively. This sort of a decomposition of weak implication-style operators looks well familiar from linear logic [5], or the even earlier work [2] by Blass. So, the above discussion of various new sorts of reduction can be in fact considered an informal introduction of the corresponding series of new recurrence operations. For this reason, and also for the (related) reason of being the only fully resource-conscious reduction, the operation $\to$ is at least as important, basic and natural as $\circ\!\!-$.

The operations $\succ$, $\circ\!\!-^{\aleph_0}$ and $\circ\!\!-$ equally enjoy the status of being conservative generalizations of Turing reduction for the interactive context. Specifically, when A and B are traditional sorts of problems such as decision problems or problems of computing a function, effective winnability of any of the three games $A \succ B$, $A \circ\!\!-^{\aleph_0} B$, $A \circ\!\!- B$ turns out to coincide with Turing reducibility of B to A, with the subtle differences between $\succ$, $\circ\!\!-^{\aleph_0}$, $\circ\!\!-$ becoming relevant only when these operators are applied to problems with higher degrees of interactivity. The same does not extend to $A \to B$ though: (even) when A and B are traditional sorts of problems, effective winnability of $A \to B$ means something properly stronger. It means existence of a Turing machine that solves B with an oracle for A where the oracle can be queried only once (while, as we know, ordinary Turing reducibility does not impose any limits on how many times the oracle can be queried). The earlier mentioned finite versions $\succ^F$ and $\circ\!\!-^F$ of weak reductions can also be seen to be conservative generalizations of Turing reduction. As for the bounded versions of weak reductions, they generalize certain proper strengthenings of Turing reduction, obtained by imposing (finite or transfinite) bounds on the number of possible queries of the oracle. Going back to our Kolmogorov complexity example, the game $\mathbf{H} \supset \mathbf{K}$ has an algorithmic winning strategy for each $\supset \in \{\succ, \circ\!\!-, \circ\!\!-^{\aleph_0}, \succ^F, \circ\!\!-^F\}$. Furthermore, with some thought and keeping in mind the known fact that the Kolmogorov complexity of n never exceeds n itself (for the exception of a finite number of "very small" n's), one can see that $\mathbf{H} \supset \mathbf{K}$ remains algorithmically solvable with either $\supset \in \{\succ^\omega, \circ\!\!-^\omega\}$ as well.

As it turns out, the logical behaviors of $\succ$, $\circ\!\!-^{\aleph_0}$ and $\circ\!\!-$ are indistinguishable when these operators are taken in isolation, and that common behavior is precisely captured by the implicative fragment $\mathbf{Int}^{\supset}$ of Heyting's intuitionistic calculus. For $\succ$ and $\circ\!\!-$, a proof of this fact was given in [12]. And the present paper extends that result to $\circ\!\!-^{\aleph_0}$ as well. As for $\to$, it turns out that its logical behavior is captured by **CL7**, which is (the Gentzen-style axiomatization of) $\mathbf{Int}^{\supset}$ with just the contraction rule deleted. In other words, **CL7** is nothing but the implicative fragment of affine logic. A proof of this result is the main technical contribution of the present article. And this is not a result that could be taken for granted. As shown in [17], affine logic in its full language, while sound, is far from being complete with respect to the semantics of computability logic. In fact, even just the $(\to, \neg)$-fragment of computability logic is not the same as the corresponding fragment of affine logic, nor does it appear to be axiomatizable in traditional proof theory.

Another way to summarize the main technical result of the present paper is to say that the set of implicative binary tautologies and their substitutional instances is precisely described by **CL7**. Here *binary tautologies* mean tautologies of classical logic where no atom occurs more than twice, and *implicative binary tautologies* are binary tautologies that contain no connectives others than $\to$. Binary tautologies and their instances have arisen in the past as a class of formulas sound and complete with respect to several natural semantics, most notably Blass's game semantics for linear logic [3], Blass's resource-conscious semantics for classical logic [4], the semantics of computability logic [6], and abstract resource semantics [10, 16]. This class of formulas has stubbornly resisted any axiomatization attempts within the framework of traditional deductive approaches and, as argued by Blass in [3], apparently this phenomenon is not quite an accident. A reasonable axiomatization for the set of binary tautologies and their instances was eventually found in [10], but it took switching to a substantially new deductive framework called *cirquent calculus* (roughly, it is sequent calculus where formulas may be shared between different sequents), indirectly corroborating Blass's thesis that binary tautologies are foreign to traditional proof theory. Against this background, the

[2] In fact, computability logic further decomposes $\to$, defining $A \to B$ as $\neg A \vee B$.

fact that the implicative fragment of that wild class can still be tamed with traditional means such as substructural sequent calculus in which **CL7** is constructed, is worth receiving our attention.

2 Logic CL7

The languages that we consider in this paper have infinitely many nonlogical propositional atoms for which we use the metavariables P, Q, and have no logical atoms. Where $\supset \in \{\rightarrow, \succ\!\!-, \circ\!\!-^{\aleph_0}, \circ\!\!-\}$, by a $\supset$**-formula** we mean a formula built from atoms and (the binary) $\supset$ in the standard way. We will be using E, F, G, H as metavariables for formulas, and Γ, Δ as metavariables for (possibly empty) multisets of formulas. As usual, we write Γ, Δ or Γ, F instead of $\Gamma \cup \Delta$ or $\Gamma \cup \{F\}$. A (two-sided) $\supset$**-sequent** is a pair $\Gamma \Rightarrow F$, where Γ is a finite multiset of $\supset$-formulas and F is a $\supset$-formula. Here Γ is said to be the **antecedent** of the sequent, and F is said to be the **succedent**.

We axiomatize **CL7** using two-sided $\rightarrow$-sequents. A ($\rightarrow$-) formula H is considered provable in this system (written **CL7** $\vdash H$) iff the empty-antecedent sequent $\Rightarrow H$ is so.

The **axioms** of **CL7** are all $\rightarrow$-sequents of the form

$$\Gamma,\ F\ \Rightarrow\ F.$$

And the system only has the following two **rules of inference**:

$$\textbf{Left} \rightarrow \qquad\qquad \textbf{Right} \rightarrow$$

$$\frac{\Gamma,\ F\ \Rightarrow\ G \qquad \Delta\ \Rightarrow\ E}{\Gamma,\ \Delta,\ E{\rightarrow}F\ \Rightarrow\ G} \qquad\qquad \frac{\Gamma,\ E\ \Rightarrow\ F}{\Gamma\ \Rightarrow\ E{\rightarrow}F}$$

We say that a formula of classical propositional logic (with $\rightarrow$-formulas here also seen as such) is **binary** iff no atom occurs in it more than twice. The concepts of being binary, tautological, true or false extend from formulas to sequents by understanding each sequent $E_1, \ldots, E_n \Rightarrow F$ as the formula $E_1 \wedge \ldots \wedge E_n \rightarrow F$. A (substitutional) **instance** of a given formula F, as usual, means the result of replacing atoms in F by any formulas, with all occurrences of the same atom being replaced by the same formula, of course.

Theorem 2.1 *For any $\rightarrow$-formula H, the following conditions are equivalent:*

(i) **CL7** $\vdash H$.

(ii) H is an instance of a binary tautology.

(iii) H is valid in computability logic, whether it be in the ordinary sense of validity or in the stronger sense of what is called "uniform validity" (see [17]).

Proof. The equivalence between (ii) and (iii) in a stronger form which is not restricted to just $\rightarrow$-formulas, has been proven in [10].[3] So, to prove the present theorem, it would be sufficient to show that (i) implies (iii) (call this *soundness*) and that (ii) implies (i) (call this *completeness*). This will be done in the following two sections. □

3 The soundness of CL7

We can rewrite **CL7** into a clearly equivalent system that uses **one-sided sequents**, here restricted to finite multisets of formulas of classical propositional logic without $\rightarrow$, where negation is applied only to atoms. This is done by rewriting each $\rightarrow$-sequent $E_1, \ldots, E_n \Rightarrow F$ as $\neg E_1, \ldots, \neg E_n, F$, and then iteratively rewriting each (sub)formula $E \rightarrow F$ as $\neg E \vee F$, each subformula $\neg(E \vee F)$ as

[3] A game-semantical soundness and completeness of the class of substitutional instances of binary tautologies was first proven with respect to Blass's game semantics in [3].

$\neg E \wedge \neg F$, each subformula $\neg(E \wedge F)$ as $\neg E \vee \neg F$ and each subformula $\neg\neg E$ as E. The axioms of the resulting system are all sequents of the form $\Gamma, \neg F, F$,[4] and the rules of inference now read as follows:

$$\textbf{Left} \rightarrow \qquad \frac{\Gamma,\ \neg F,\ G \qquad\qquad \Delta,\ E}{\Gamma,\ \Delta,\ E \wedge \neg F,\ G}$$

$$\textbf{Right} \rightarrow \qquad \frac{\Gamma,\ \neg E,\ F}{\Gamma,\ \neg E \vee F}$$

Among several equivalent axiomatizations of the (multiplicative fragment of the) well known *affine logic* is the one that uses one-sided sequents in our present sense. It has the same axiom scheme $\Gamma, \neg F, F$. And the above Right $\rightarrow$ and Left $\rightarrow$ rules are special cases of the $\vee$-introduction and $\wedge$-introduction rules of that system, respectively, where $\wedge, \vee$ are seen as multiplicatives.[5] Thus, understanding $E \rightarrow F$ as an abbreviation of $\neg E \vee F$, affine logic proves every $\rightarrow$-formula provable in **CL7**. But, as proven in [17], affine logic is sound with respect to the semantics computability logic, and the latter sees no difference between $E \rightarrow F$ and $\neg E \vee F$. So, clause (i) of Theorem 2.1 implies clause (iii), as desired.

4 The completeness of CL7

We define the **head** of a $\rightarrow$-formula as follows:

- Every atom is its own head.
- The head of $E \rightarrow F$ is that of F.

In other words, the head of a formula is the atom with the rightmost occurrence in the formula — the (unique) occurrence that is not in the antecedent of any subformula.

Consider any binary $\rightarrow$-sequent $\Gamma \Rightarrow F$. We define the **relevant formulas** of this sequent to be the elements of the smallest set S such that:

- Every formula of Γ whose head occurs in F is in S.
- Every formula of Γ whose head occurs in some element of S is also in S.

The formulas of Γ that are not relevant will be said to be **irrelevant**.

Lemma 4.1 *Assume $\Gamma \Rightarrow F$ is a binary tautological $\rightarrow$-sequent, and Δ is the result of deleting from Γ all irrelevant formulas of $\Gamma \Rightarrow F$. Then the sequent $\Delta \Rightarrow F$ is also tautological (and, of course, remains binary).*

Proof. Let Γ, Δ, F be as above. In what follows, by a "relevant formula" we always mean a relevant formula of $\Gamma \Rightarrow F$. Similarly for "irrelevant", "antecedent", "succedent".

Suppose that $\Delta \Rightarrow F$ is not tautological. Consider a truth assignment that makes it false, i.e., makes Δ true and F false. Extend it to all formulas of Γ by stipulating that, if an atom does not occur in $\Delta \Rightarrow F$, it is true. Obviously the head of every irrelevant formula is true under this extended assignment and hence every irrelevant formula is true. All relevant formulas of the antecedent also remain true. And the succedent remains false. So, $\Gamma \Rightarrow F$ is false and hence non-tautological. $\Box$

[4] Of course, it does not matter whether here and later we write Γ or $\neg\Gamma$, with $\neg\Gamma$ meaning the multiset of the negations of the elements of Γ.

[5] In fact, writing E instead of $\neg E$, Right $\rightarrow$ is simply the same as the $\vee$-introduction rule of affine logic.

Lemma 4.2 *Assume $\Gamma \Rightarrow E$ and $\Gamma \Rightarrow F$ are binary sequents, where E and F do not share any atoms. Then the sets of relevant formulas of the two sequents are disjoint.*

Proof. Assume the conditions of the lemma. Consider an arbitrary relevant formula G of $\Gamma \Rightarrow E$. Let P be the head of G. If the reason for G's relevance is that P occurs in E, then (as E and F share no atoms) P does not occur in F, nor does it occur in any formula of Γ other than G because of the binarity of the sequent. This, by the definition of relevance, means that G is not a relevant formula of $\Gamma \Rightarrow F$.

Suppose now the reason for G's being a relevant formula of $\Gamma \Rightarrow E$ is that P occurs in some relevant formula H of $\Gamma \Rightarrow E$. The relevance of H has thus been established earlier than that of G and hence, by the induction hypothesis, H is not a relevant formula of $\Gamma \Rightarrow F$. But, in view of binarity, the only two places where P occurs (whether it be within $\Gamma \Rightarrow E$ or $\Gamma \Rightarrow F$) are in G and H. Hence G cannot be a relevant formula of $\Gamma \Rightarrow F$. $\Box$

Lemma 4.3 CL7 *proves every binary tautological $\rightarrow$-sequent.*

Proof. Consider an arbitrary binary tautological sequent. We may assume that its succedent is an atom P, for otherwise, if the succedent is $E \rightarrow F$, move E to the antecedent of the sequent, and repeat the same until the succedent has become atomic; in view of the presence of Right $\rightarrow$ in **CL7**, provability of the resulting sequent implies provability of the original one.

If P is one of the formulas of the antecedent, then the sequent we deal with is an axiom and thus **CL7** proves it.

Otherwise, the antecedent should contain a formula $E \rightarrow F$ whose head is P, or else the sequent could be falsified by the truth assignment which makes P false and makes all other atoms true. Thus, the sequent we are talking about looks like $\Gamma, E \rightarrow F \Rightarrow P$, where P occurs in F and hence occurs in neither E nor Γ, as the sequent is binary. Obviously the tautologicity of this sequent implies the tautologicity of $\Gamma, F \Rightarrow P$. Since E does not contain P, the tautologicity of $\Gamma, E \rightarrow F \Rightarrow P$ also implies the tautologicity of $\Gamma \Rightarrow E$. Indeed, assume that some truth assignment falsifies $\Gamma \Rightarrow E$. Extend that assignment to all atoms of $\Gamma, E \rightarrow F \Rightarrow P$ in such a way that it makes P false. Obviously such an extended assignment falsifies $\Gamma, E \rightarrow F \Rightarrow P$, contradicting our assumption that this sequent is tautological. Thus, $\Gamma, F \Rightarrow P$ and $\Gamma \Rightarrow E$ are binary tautological sequents, and their succedents do not share any atoms. Let Γ_1 and Γ_2 be the submultisets of Γ consisting of all relevant formulas of $\Gamma, F \Rightarrow P$ and $\Gamma \Rightarrow E$, respectively. By Lemma 4.2, Γ_1 and Γ_2 are disjoint. Also, by Lemma 4.1, $\Gamma_1, F \Rightarrow P$ and $\Gamma_2 \Rightarrow E$ are tautological. Hence, by the induction hypothesis (where induction is on the number of connectives occurring the sequent), these two sequents are provable. Then, by Left $\rightarrow$, the sequent $\Gamma_1, \Gamma_2, E \rightarrow F \Rightarrow P$ is also provable. This can be easily seen to imply the provability of the original sequent $\Gamma, E \rightarrow F \Rightarrow P$, as **CL7** is obviously closed under the weakening rule "from $\Delta \Rightarrow G$ conclude $\Delta, H \Rightarrow G$".[6] $\Box$

In view of the evident fact that **CL7** is closed under substitution of atoms by whatever formulas, Lemma 4.3 immediately implies the desired conclusion that, whenever H is a $\rightarrow$-formula which is an instance of some binary tautology, H is provable in **CL7**.

5 The three versions of weak reduction

As noted in Section 1, the three weak reduction operations $\succ$, $\circ\!\!-^{\aleph_0}$ and $\circ\!\!-$ can be defined in terms of $\rightarrow$ and the corresponding three *recurrence* operations $\curlywedge, \circ^{\downarrow\aleph_0}, \circ^{\downarrow}$ by

$$A \succ B =_{def} \curlywedge A \rightarrow B; \quad A \circ\!\!-^{\aleph_0} B =_{def} \circ^{\downarrow\aleph_0} A \rightarrow B; \quad A \circ\!\!- B =_{def} \circ^{\downarrow} A \rightarrow B.$$

[6] In the present version of **CL7**, weakening is "hidden" in axioms. Alternatively, we could have chosen the axioms of **CL7** to be just $F \Rightarrow F$, with weakening explicitly stipulated as one of the inference rules. It is known that either choice yields the same set of provable formulas, whether it be classical, affine or intuitionistic logic.

(Recurrences have the highest precedence, so $\curlywedge A \rightarrow B$ should be read as $(\curlywedge A) \rightarrow B$, and similarly for $\mathbin{\rotatebox{180}{$\curlywedge$}}^{\aleph_0}, \mathbin{\rotatebox{180}{$\curlywedge$}}$.) We refer to $\curlywedge$ as **parallel recurrence**, and refer to $\mathbin{\rotatebox{180}{$\curlywedge$}}^{\aleph_0}$ and $\mathbin{\rotatebox{180}{$\curlywedge$}}$ as **branching recurrences**. Namely, $\mathbin{\rotatebox{180}{$\curlywedge$}}^{\aleph_0}$ can be called **countable** branching recurrence, and $\mathbin{\rotatebox{180}{$\curlywedge$}}$ called **uncountable** branching recurrence. $\mathbin{\rotatebox{180}{$\curlywedge$}}$ and $\curlywedge$ have been defined in some earlier literature on computability logic (see, e.g., [17]). On the other hand, $\mathbin{\rotatebox{180}{$\curlywedge$}}^{\aleph_0}$, as a full-fledged citizen of computability logic, is first officially introduced in the present paper (see also "Historical remarks" at the end of this section).

Let us start with taking a closer (than done in Section 1) intuitive look at how $\mathbin{\rotatebox{180}{$\curlywedge$}}$ and $\curlywedge$ compare. Imagine a computer that has a program successfully playing *Chess*. The resource that such a computer provides is obviously stronger than just *Chess*: a reasonable operating system would allow to simultaneously run as many parallel sessions of *Chess* as the user needs, while *Chess*, as such, only assumes a single play. This is what is captured by the parallel recurrence $\curlywedge$*Chess*. A more advanced operating system, however, would in addition also make it possible to branch/replicate each particular stage of each particular session, i.e. create any number of "copies" of any already reached position of the multiple parallel plays of *Chess*, thus giving the user the possibility to try different continuations from the same position. What corresponds to this intuition is the branching recurrence $\mathbin{\rotatebox{180}{$\curlywedge$}}$*Chess*.

As noted earlier, the user of the resource $\mathbin{\rotatebox{180}{$\curlywedge$}} A$ does not have to restart A from the very beginning every time it wants to reuse it; rather, it is (essentially) allowed to backtrack to any of the previous — not necessarily starting — positions and try a new continuation from there, thus depriving the adversary of the possibility to reconsider the moves it has already made in that position. This is in fact the type of reusage every purely software resource allows or would allow in the presence of an advanced operating system and unlimited memory: one can start running process A; then fork it at any stage thus creating two threads that have a common past but possibly diverging futures (with the possibility to treat one of the threads as a "backup copy" and preserve it for backtracking purposes); then further fork any of the branches at any time; and so on. The less flexible type of reusage of A assumed by $\curlywedge A$, on the other hand, is closer to what infinitely many autonomous physical resources would naturally offer, such as an unlimited number of independently acting robots each performing task A, or an unlimited number of computers with limited memories, each one only capable of and responsible for running a single thread of process A. Here the effect of replicating/forking an advanced stage of A cannot be achieved unless, by good luck, there are two identical copies of the stage, meaning that the corresponding two robots or computers have so far acted in precisely the same ways.

The difference between the countable ($\mathbin{\rotatebox{180}{$\curlywedge$}}^{\aleph_0}$) and uncountable ($\mathbin{\rotatebox{180}{$\curlywedge$}}$) versions of branching recurrence appears to be much more subtle than the difference between the parallel ($\curlywedge$) and branching ($\mathbin{\rotatebox{180}{$\curlywedge$}}$ or $\mathbin{\rotatebox{180}{$\curlywedge$}}^{\aleph_0}$) sorts of recurrence. In fact, the above intuitive-level discussion of $\curlywedge$ vs. $\mathbin{\rotatebox{180}{$\curlywedge$}}$ is just as valid for $\curlywedge$ vs. $\mathbin{\rotatebox{180}{$\curlywedge$}}^{\aleph_0}$. Yet, $\mathbin{\rotatebox{180}{$\curlywedge$}}$ and $\mathbin{\rotatebox{180}{$\curlywedge$}}^{\aleph_0}$ turn out to induce dramatically different logics, even if those logics coincide when $\circ\!\!-\!\!\!\!\rightarrow^{\aleph_0}$ or $\circ\!\!-\!\!\!\!\rightarrow$ (or $\succ$) is the only connective in the logical vocabulary. The following is an example of a principle which could be shown to be valid with $\mathbin{\rotatebox{180}{$\curlywedge$}}$ but invalid with $\mathbin{\rotatebox{180}{$\curlywedge$}}^{\aleph_0}$ as well as with $\curlywedge$:

$$\mathbin{\rotatebox{180}{$\curlywedge$}} \mathbin{\rotatebox{180}{$\curlyvee$}} P \rightarrow \mathbin{\rotatebox{180}{$\curlyvee$}} \mathbin{\rotatebox{180}{$\curlywedge$}} P$$

($\mathbin{\rotatebox{180}{$\curlyvee$}}$ abbreviates $\neg \mathbin{\rotatebox{180}{$\curlywedge$}} \neg$, where $\neg$ is the "role switch" operation). And, as we started discussing differences between the principles validated by the different sorts of recurrences, here comes an example of a principle which can be shown to be valid with $\curlywedge$ but invalid with either $\mathbin{\rotatebox{180}{$\curlywedge$}}$ or $\mathbin{\rotatebox{180}{$\curlywedge$}}^{\aleph_0}$:

$$P \wedge \curlywedge (P \rightarrow Q \wedge P) \rightarrow \curlywedge Q$$

($\wedge$, called *parallel conjunction*, is a computability-logic counterpart of the tensor of linear logic. $A \wedge B$ means a parallel play of A and B, where $\top$ has to win in both plays to be the winner in the overall game). These are just isolated examples, and finding a systematic deductive characterization of all valid principles that involve recurrence operations remains a great challenge in computability logic.

Before we move to more examples illustrating differences between $\curlywedge, \mathbin{\rotatebox{180}{$\curlywedge$}}^{\aleph_0}$ and $\mathbin{\rotatebox{180}{$\curlywedge$}}$, it would be a good idea to first define the three operations under question.

Formally, $⅄A$ is defined as the infinite conjunction $A \wedge A \wedge A \wedge \ldots$, where $A_0 \wedge A_1 \wedge A_2 \wedge \ldots$ is a straightforward generalization of the just-mentioned parallel conjunction $\wedge$ operation from the binary case to the infinite case.

Defining the branching recurrences takes more work. In semiformal terms, a play of $⅄A$ starts as an ordinary play of game A. At any time, however, player $\bot$ is allowed to make a "replicative move", which creates two copies of the current position Φ of A. From that point on, the game turns into two games played in parallel, each continuing from position Φ. We use the bits 0 and 1 to denote those two threads, that — using our earlier words — have a common past (position Φ) but possibly diverging futures. Again, at any time, $\bot$ can further branch either thread, creating two copies of the current position in that thread. If thread 0 was branched, the resulting two threads will be denoted by 00 and 01; and if the branched thread was 1, then the resulting threads will be denoted by 10 and 11. And so on: at any time, $\bot$ may split any of the existing threads w into two threads $w0$ and $w1$. Each thread in the eventual run of the game will be thus denoted by a (possibly infinite) bit string. The game is considered won by $\top$ if it wins A in each of the threads; otherwise the winner is $\bot$.

To each infinite bit string w may thus correspond a separate run of A in thread (represented by) w and, as there are uncountably many infinite bit strings, uncountably many parallel runs of A may be generated when playing $⅄A$ up. Let us call a bit string w **essentially finite** if it contains only a finite number of 1s; otherwise we say that w is **essentially infinite**. We extend these terms from bit strings to the corresponding threads in the play of $⅄A$. The definition of $⅄A$ thus requires from $\top$ to win A in all — whether they be essentially finite or essentially infinite — threads. All it takes to turn that definition into a definition of $⅄^{\aleph_0}$ is to relax that requirement and, when determining the winner, only look at essentially finite threads. Since there are only countably many essentially finite bit strings, only countably many runs of A are generated — more precisely, only countably many runs of A are of relevance — in $⅄^{\aleph_0}A$. This completes our semiformal definition/explanation of $⅄^{\aleph_0}$.

In fully formal terms, both $⅄A$ and $⅄^{\aleph_0}A$ have the same structures (**Lr** components). There are two types of legal moves in (legal) positions of either game: (1) replicative and (2) non-replicative. To define these, let us agree that by an *active node*[7] of a position Φ we mean a bit string w such that w is either empty,[8] or else is $u0$ or $u1$ for some bit string u such that Φ contains the move u:. A replicative move can only be made by (is only legal for) $\bot$, and such a move in a given position Φ should be w:, where w is an active node of Φ and Φ does not already contain the same move w:.[9] As for non-replicative moves, they can be made by either player. Such a move by a player $\wp$ in a given position Φ should be $w.\alpha$, where w is an active node of Φ and α is a move such that, for any infinite bit string v, α is a legal move by $\wp$ in position $\Phi^{\preceq wv}$ of A.[10] Here and later, for a run Θ and bit string x, $\Theta^{\preceq x}$ means the result of deleting from Θ all moves except those that look like $u.\beta$ for some initial segment u of x, and then further deleting the prefix "u." from such moves.[11] As for the contents (the **Wn** components) of these games, a legal run Γ of $⅄A$ is considered won by $\top$ iff, for every infinite bit string v, $\Gamma^{\preceq v}$ is a $\top$-won run of A. And a legal run Γ of $⅄^{\aleph_0}A$ is considered won by $\top$ iff, for every infinite but essentially finite bit string v, $\Gamma^{\preceq v}$ is a $\top$-won run of A. This completes our definition of $⅄$ and $⅄^{\aleph_0}$.

As we just saw, the definition of $⅄^{\aleph_0}$ is obtained from the definition of $⅄$ by merely inserting the words "but essentially finite". But, again, trying to analyze the rather technical definition given in the above paragraph may not be a good idea for a reader of this paper. Relying, instead, on

[7] Intuitively, an active node is (the name of) an already existing thread of a play over A.

[8] Intuitively, the empty string is the name/address of the initial thread; all other threads will be descendants of that thread.

[9] The intuitive meaning of move w: is splitting thread w into $w0$ and $w1$, thus "activating" these two new nodes/threads.

[10] The intuitive meaning of such a move $w.\alpha$ is making move α in thread w and all of its (current or future) descendants.

[11] Intuitively, $\Theta^{\preceq x}$ is the run of A that has been played in thread x, if such a thread exists (has been generated); otherwise, $\Theta^{\preceq x}$ is the run of A that has been played in (the unique) existing thread which (whose name, that is) is some initial segment of x.

the informal explanations that we provided should be sufficient.

To see the distance between $\mathbin{⫰}$ and $\mathbin{⫰}^{\aleph_0}$, following Vereshchagin [20], let us consider any set S of natural numbers (identified with their decimal representations), such that S is not recursively enumerable. Let A be the game where only $\top$ has legal moves, each legal move being a(ny) natural number. A given run of this game is considered won by $\top$ iff the set of the moves it makes in it equals S. In other words, $\top$ wins iff it enumerates S. Now let us look at the games $⫯A$ and $⫯^{\aleph_0}A$, where $⫯ = \neg\mathbin{⫰}\neg$ and $⫯^{\aleph_0} = \neg\mathbin{⫰}^{\aleph_0}\neg$. That is, $⫯A$ is the same as $\mathbin{⫰}A$, only here it is $\top$ rather than $\bot$ who can create new threads (make replicative moves), and whose adversary needs to win A in each of the threads to be the winner in the overall game. Similarly for $⫯^{\aleph_0}$. Obviously $\top$ has an effective winning strategy for $⫯A$, consisting in enumerating all of the $2^{\aleph_0}$ sets of natural numbers, one per each of the $2^{\aleph_0}$ threads that it may create in $⫯A$. On the other hand, $\top$ does not have an effective winning strategy for $⫯^{\aleph_0}A$. Otherwise, one would be able to recursively enumerate S by selecting the (essentially finite) bit string w representing a winning thread, and then listing the moves made in that thread.

Our discussion would not be complete without also seeing a specific example illustrating the distance between the parallel and branching versions of recurrence. The game $⅄A$, which is a dual of $⅂A$ in the same sense as $⫯A$ is a dual of $\mathbin{⫰}A$, is defined as $A \vee A \vee A \vee \ldots$. This can be thought of as a parallel play of game A on infinitely many boards: #0, #1, #2, ... $\bot$ wins it iff it wins A on each of the boards. Where $f(x)$ is a total function from natural numbers to natural numbers, $\bigsqcap x\bigsqcup y\big(y = f(x)\big)$ denotes a game every legal run of which consists of (at most) two moves.[12] The first move is by $\bot$, and the move is an arbitrary number m. The second move is by $\top$, who should name a number n. $\top$ wins iff n equals $f(m)$. $\top$ has an effective winning strategy that works for both $⫯\bigsqcap x\bigsqcup y\big(y = f(x)\big)$ and $⫯^{\aleph_0}\bigsqcap x\bigsqcup y\big(y = f(x)\big)$. It consists in waiting till the adversary makes a move m, after which $\top$ creates infinitely (but countably) many threads, and tries all possible responses — all possible values for n, that is — in those threads, one response per thread. Similarly, where $B(x)$ is a predicate, $\bigsqcap x(\neg B(x) \sqcup B(x))$ denotes a game where the first move (again), consisting of choosing a number m, is by $\bot$. The second move is by $\top$, who should choose between 0 and 1. $\top$ wins iff $B(m)$ is false and 0 was chosen, or $B(m)$ is true and 1 was chosen. $\top$'s effective winning strategy for both $⫯\bigsqcap x(\neg B(x) \sqcup B(x))$ and $⫯^{\aleph_0}\bigsqcap x(\neg B(x) \sqcup B(x))$ is that it waits till the adversary makes a move m, after which $\top$ creates two threads, making move 0 in one thread and move 1 in the other thread. The same trick, however, fails with $⅄\bigsqcap x(\neg B(x) \sqcup B(x))$. For example, it fails when $\bot$ chooses $m = 0$ on board #0, $m = 1$ on board #1, $m = 2$ on board #2, etc. Let us call this strategy of $\bot$ the *diversifying strategy*. Now, for any effective strategy $\mathcal{M}$ of $\top$, using diagonalization, we can construct a particular predicate $B(x)$ such that $\top$ loses $⅄\bigsqcap x(\neg B(x) \sqcup B(x))$ against the diversifying strategy. Namely, we can define $B(i)$ (any i) to be true if $\mathcal{M}$ makes the move 0 on board $\#i$ when playing against the diversifying strategy, and false otherwise. This guarantees that $\mathcal{M}$'s all responses to the adversary's moves are "wrong". A similar idea could be employed in showing that, for an appropriately selected f, the problem $⅄\bigsqcap x\bigsqcup y\big(y = f(x)\big)$ has no algorithmic solution.

Historical remarks. Blass [2] was apparently the first to consider an operator in the style of the exponential operator ! of linear logic. He called it the *repetition operator* R. The game-semantical context in which R was introduced was limited compared with the context that computability logic operates in. The main contextual difference is that Blass's games are *strict*, meaning games where in each position only one player may have (legal) moves. Computability logic, on the other hand, deals with the already mentioned more general type of *static games*. As opposed to strict games, static games are *free*, in the sense that generally both players may have legal moves in a given position. Furthermore, the recurrence operations (as well as the non-recurrence parallel operations $\wedge, \vee, \bigwedge, \bigvee$) of computability logic generate properly free games even when applied to strict games, while Blass's operations, of course, preserve the strict property of games. However,

[12] In computability logic, $\bigsqcap$ is called *choice universal quantifier*, and $\bigsqcup$ called *choice existential quantifier*. The smaller versions $\sqcap$ and $\sqcup$ of the same symbols stand for what are called *choice conjunction* and *choice disjunction*, respectively. The choice operators of computability logic are reminiscent of the additive operators of linear logic.

if we disregard this difference and try to bring Blass's games and static games to some reasonable common denominator, Blass's repetition operation R would apparently translate (whatever "translate" should precisely mean here) into $\mathbin{⫰}^{\aleph_0}$. The reason why it would not translate into $\curlywedge$ is that R is a branching operation in the proper sense, allowing effects such as backtracking. And the reason why R would be less than an adequate counterpart of $⫰$ is that $⫰A$ allows $\bot$ to try a continuum of different runs of A, while that quantity is automatically limited to $\aleph_0$ in RA (and artificially limited to $\aleph_0$ in $\mathbin{⫰}^{\aleph_0}A$, as we saw from the definition).

6 Implicative intuitionistic logic

Where $\supset$ is one of the operators $\succ$, $\circ\!\!-^{\aleph_0}$ or $\circ\!\!-$, a Gentzen-style axiomatization of the corresponding *implicative* (fragment of) *intuitionistic logic*, denoted by $\mathbf{Int}^{\supset}$, is **CL7** — only with $\supset$-sequents instead of $\rightarrow$-sequents, of course — plus the following single additional rule

Contraction

$$\frac{\Gamma,\ E,\ E\ \Rightarrow\ F}{\Gamma,\ E\ \Rightarrow\ F}$$

Alternatively, $\mathbf{Int}^{\supset}$ could be chosen to be formulated exactly as **CL7**, with the only difference that the antecedents of sequents in $\mathbf{Int}^{\supset}$ are seen as sets rather than multisets of formulas, which eliminates the need for explicitly stating contraction as an inference rule.

Theorem 6.1 *Let $\supset$ be any one of the operators $\succ$, $\circ\!\!-^{\aleph_0}$ or $\circ\!\!-$. For any $\supset$-formula H, the following conditions are equivalent:*

(i) $\mathbf{Int}^{\supset} \vdash H$.

(ii) H is valid in computability logic, whether it be in the ordinary sense of validity or in the sense of uniform validity.

Proof. For $\mathbf{Int}^{\succ}$ and $\mathbf{Int}^{\circ\!-}$, this theorem was officially established in [12]. As for $\mathbf{Int}^{\circ\!-^{\aleph_0}}$, as it happens, the proof of the soundness and completeness of $\mathbf{Int}^{\circ\!-}$ given in [12], in fact, is also a proof of the soundness and completeness of $\mathbf{Int}^{\circ\!-^{\aleph_0}}$: a simple re-reading of that proof reveals that virtually no step in it relies on the fact that we deal with the uncountable rather than the countable version of reduction. □

Historical remarks and further discussions. The above-mentioned result of [12] for $\mathbf{Int}^{\circ\!-}$ was further strengthened in [15], where soundness and completeness (with respect to the semantics of computability logic) was proven for the full propositional fragment of intuitionistic logic. With only one or two months' delay, Vereshchagin [20] came up with an alternative and shorter proof of the same result. It should be noted, however, that in his work Vereshchagin modified the "canonical" definitions of computability logic quite a bit, which essentially resulted in interpreting intuitionistic implication as $\circ\!\!-^{\aleph_0}$ rather than $\circ\!\!-$. Moreover, in an attempt to simplify things, Vereshchagin further limited games to strict ones, essentially defining the $\mathbin{⫰}^{\aleph_0}$ component of $\circ\!\!-^{\aleph_0}$ as something closer to Blass's repetition operator R than to $\mathbin{⫰}^{\aleph_0}$ in our present precise sense. In view of (and despite) the above-said, Vereshchagin's proof, with certain technical adjustments, can be considered an alternative proof of the $\mathbf{Int}^{\circ\!-^{\aleph_0}}$ part of Theorem 6.1.

The soundness proof for $\mathbf{Int}^{\circ\!-}$ found in [12], in fact, can be dramatically simplified when we are concerned with $\circ\!\!-^{\aleph_0}$ rather than $\circ\!\!-$. Specifically, a lemma on which the soundness proof given in [12] (as well as similar proofs given in [2] and [20]) relies is about the validity of the principle $⫰A \rightarrow ⫰⫰A$. And a strict proof of that lemma, presented in [17], takes several pages. On the other

hand, a proof of the validity of the same principle for $\downarrow\!\!\circ^{\aleph_0}$ instead of $\downarrow\!\!\circ$ would not take more than just a paragraph, as it did (for Blass's or Vereshchagin's versions of $\downarrow\!\!\circ^{\aleph_0}$) in [2] or [20].

The above-said also applies to the proof of the soundness and completeness of the full propositional intuitionistic logic given in [15]. That proof officially is for the case when the intuitionistic implication is read as $\circ\!\!-$. However, the same proof is just as good for $\circ\!\!-^{\aleph_0}$ as well. Similarly, Vereshchagin's [20] completeness proof can be adapted to either interpretation $\circ\!\!-$, $\circ\!\!-^{\aleph_0}$ of intuitionistic implication. The same cannot be said about Vereshchagin's soundness proof though: as noted above, proving soundness when intuitionistic implication is read as $\circ\!\!-$ rather than $\circ\!\!-^{\aleph_0}$ takes considerably greater efforts.

Finally, for reasons similar to the above, the soundness proof for the full first-order intuitionistic calculus given in [14], is equally good for either reading $\circ\!\!-$, $\circ\!\!-^{\aleph_0}$ of intuitionistic implication.

7 Conclusion

Computability logic is a semantically conceived approach with the ambition to be "a formal theory of computability in the same sense as classical logic is a formal theory of truth" ([7]). As such, it does not yet have a sufficiently developed syntax, and among the main current objectives of computability logic as a research program is to find axiomatizations for various natural fragments of the set of formulas validated by its semantics. The language of computability logic, with logical operators standing for operations on computational problems, is very rich and, in fact, open-ended. Identifying the most natural and potentially useful new operators to be included in it is another direction on which efforts within the project continue to be focused.

The present paper contributes to both of the above directions. Within the second direction, it officially introduces the Blass-style ([2, 3]) *countable recurrence* operator $\downarrow\!\!\circ^{\aleph_0}$ and the associated reduction operator $\circ\!\!-^{\aleph_0}$. It also outlines the idea of finite and bounded versions of recurrence operators together with the associated reduction operators. The three other main reduction operators $\rightarrow$, $\succ\!\!-$ and $\circ\!\!-$ studied in the paper have been introduced earlier. The variety of reduction operators captures various flavors of our intuition of algorithmically reducing one computational problem to another, with $\rightarrow$ being the strongest form of reduction and $\circ\!\!-$ being the weakest form.

Within the first direction, this paper establishes two results. According to one result, a greater part of which was established earlier, the logic induced by each of $\succ\!\!-$, $\circ\!\!-^{\aleph_0}$, $\circ\!\!-$ is exactly the implicative fragment of Heyting's intuitionistic calculus. And, according to the other theorem, the logic induced by $\rightarrow$ is the same calculus but with the contraction rule removed. The philosophical summary of these results is that, despite the significant semantical varieties within the basic group of reduction operations, they (when taken in isolation) only generate two kinds of logical behavior, depending on whether resource/antecedent reusage is allowed ($\succ\!\!-$, $\circ\!\!-^{\aleph_0}$, $\circ\!\!-$) or not ($\rightarrow$). Syntactically, those two behaviors are precisely accounted for by the mere presence or absence of contraction in Gentzen-style axiomatizations.

References

[1] S. Abramsky and R. Jagadeesan. *Games and full completeness for multiplicative linear logic.* **Journal of Symbolic Logic** 59 (2) (1994), pp. 543-574.

[2] A. Blass. *Degrees of indeterminacy of games.* **Fundamenta Mathematicae** 77 (1972), pp. 151-166.

[3] A. Blass. *A game semantics for linear logic.* **Annals of Pure and Applied Logic** 56 (1992), pp. 183-220.

[4] A. Blass. *Resource consciousness in classical logic.* In: **Games, Logic, and Constructive Sets** (Proceedings of LLC9, the 9th conference on Logic, Language, and Computation, held at CSLI). G.Mints and R.Muskens, eds. (2003) 61-74.

[5] J.Y. Girard. *Linear logic*. **Theoretical Computer Science** 50 (1987), pp. 1-102.

[6] G. Japaridze. *Introduction to computability logic*. **Annals of Pure and Applied Logic** 123 (2003), pp. 1-99.

[7] G. Japaridze. *Propositional computability logic I*. **ACM Transactions on Computational Logic** 7 (2006), No.2, pp. 302-330.

[8] G. Japaridze. *Propositional computability logic II*. **ACM Transactions on Computational Logic** 7 (2006), No.2, pp. 331-362.

[9] G. Japaridze. *From truth to computability I*. **Theoretical Computer Science** 357 (2006), pp. 100-135.

[10] G. Japaridze. *Introduction to cirquent calculus and abstract resource semantics*. **Journal of Logic and Computation** 16 (2006), pp. 489-532.

[11] G. Japaridze. *Computability logic: a formal theory of interaction*. In: **Interactive Computation: The New Paradigm**. D. Goldin, S. Smolka and P. Wegner, eds. Springer Verlag, Berlin, 2006, pp. 183-223.

[12] G. Japaridze. *The logic of interactive Turing reduction*. **Journal of Symbolic Logic** 72 (2007), No.1, pp. 243-276.

[13] G. Japaridze. *From truth to computability II*. **Theoretical Computer Science** 379 (2007), pp. 20-52.

[14] G. Japaridze. *Intuitionistic computability logic*. **Acta Cybernetica** 18 (2007), No.1, pp. 77–113.

[15] G. Japaridze. *The intuitionistic fragment of computability logic at the propositional level*. **Annals of Pure and Applied Logic** 147 (2007), pp.187-227.

[16] G. Japaridze. *Cirquent calculus deepened*. **Journal of Logic and Computation** Advance Access published online on July 21, 2008. doi:10.1093/logcom/exn019

[17] G. Japaridze. *In the beginning was game semantics*. In: **Games: Unifying Logic, Language and Philosophy**. O. Majer, A.-V. Pietarinen and T. Tulenheimo, eds. Springer Verlag, Berlin (to appear). Preprint is available at http://arxiv.org/abs/cs.LO/0507045

[18] G. Japaridze. *Sequential operators in computability logic*. Preprint is available at http://arxiv.org/abs/0712.1345

[19] G. Japaridze. *Towards applied theories based on computability logic*. Preprint is available at http://arxiv.org/abs/0805.3521

[20] N. Vereshchagin. *Japaridze's computability logic and intuitionistic propositional calculus*. Moscow State University preprint (Russian), 2006. Available at http://lpcs.math.msu.su/∼ver/papers/japaridze.ps